Homework 8: User-level threads

제출 마감: 2018. 11. 22. 11:00 am

제출처: 오준택 조교 (na94jun@gmail.com)

Switching threads

Piazza 홈페이지에서 uthread.c 와 uthread_switch.S 파일을 다운 받아 xv6 디렉토리에 추가하라. Makefile을 열고 하기 문장을 _forkret 아래에 추가하라. (하기 문장 들여쓰기는 space가 아닌 tab이다.)

```
_uthread: uthread.o uthread_switch.o
	$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o _uthread
uthread.o uthread_switch.o $(ULIB)
	$(OBJDUMP) -S _uthread > uthread.asm
```

Makefile의 UPROGS 선언 밑에 _thread를 추가해 user program으로 등록하라.

xv6를 실행시키고 shell에서 uthread 프로그램을 실행하라. page fault error가 출력된 것을 볼 수 있을 것이다. 본 과제는 page fault error를 제거하기 위해 thread_swtich.S 파일을 완성하는 것이다. 과제를 성공적으로 완료한 후, uthread 프로그램을 실행하면 하기 문장 과 같이 출력될 것이다.
(*xv6를 동작시킬 때 중요한 점은 CPUS option을 아래와 같이, CPUS=1로 추가하는 것이다. 이 옵션은 xv6 가상 머신의 CPU 개수를 1개로 고정하는 것이다):

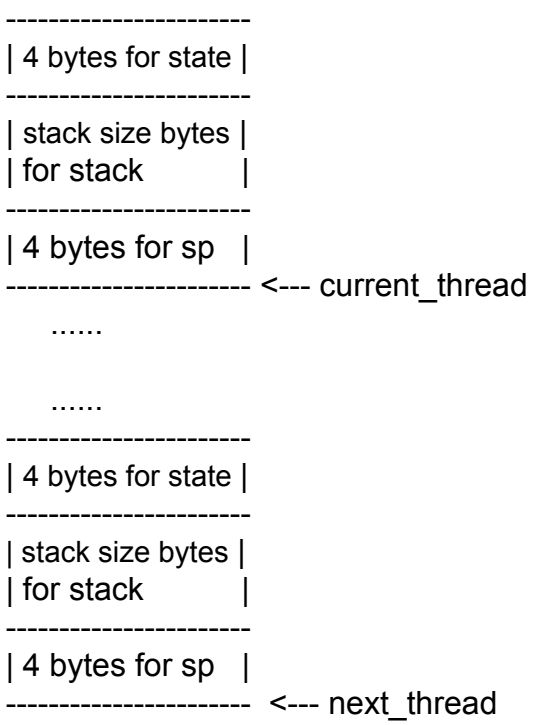
```
~/classes/6828/xv6$ make CPUS=1 qemu-nox

dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes transferred in 0.037167 secs (137756344
bytes/sec)
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes transferred in 0.000026 secs (19701685 bytes/sec)
dd if=kernel of=xv6.img seek=1 conv=notrunc
```

```
307+1 records in
307+1 records out
157319 bytes transferred in 0.003590 secs (43820143 bytes/sec)
qemu -nographic -hdb fs.img xv6.img -smp 1 -m 512
Could not open option rom 'sgabios.bin': No such file or
directory
xv6...
cpu0: starting
init: starting sh
$ uthread
my thread running
my thread 0x2A30
my thread running
my thread 0x4A40
my thread 0x2A30
my thread 0x4A40
my thread 0x2A30
my thread 0x4A40
....
```

uthread 프로그램은 thread 두개를 생성하고 번갈아가며 thread를 실행시키는 프로그램이다. 각 thread의 동작은 “my thread…” 문장을 출력하고 CPU 자원을 반환하는 yield 함수를 호출하는 것이다.

위 결과를 출력하려면, thread_switch.S 파일을 완성해야 한다. 그 전에, uthread.c 파일 내 thread_switch 함수가 어떻게 사용되고 있는지를 이해해야 한다. uthread.c는 current_thread 와 next_thread 두개의 전역 변수들을 갖고있다. 각 변수들은 thread 구조체를 가리킨다. Thread 구조체는 thread를 위한 stack 과 stack pointer를 가진다. uthread_switch의 작업은 current_thread가 가리키는 구조체에 현재 스레드 상태를 저장하고 next_thread의 상태를 복원하는 것이다. 그리고 current_thread를 next_thread가 가리키던 thread를 가리키게 하여 thread_switch가 리턴될 때 next_thread가 가리키던 thread가 실행되고, current_thread가 되도록 하라. thread_switch는 assembly 명령어 중 pushal 과 popal을 이용해 x86 register들 8개를 모두 저장하고 복구한다. thread_create는 8개의 register들을 새로운 thread stack에 넣는 동작을 모사한다.

thread_switch에서 assembly를 작성하기 위해 C compiler가 thread 구조체를 메모리에서 어떻게 관리하는지 알 필요가 있다. thread 구조체의 memory layout은 다음과 같다 :



current_thread가 가리키는 구조체의 sp 필드에 접근하려면 다음과 같이 어셈블리를 작성해야한다:

```
movl current_thread, %eax
movl %esp, (%eax)
```

위 문장은 current_thread 구조체 내의 stack pointer에 현재 esp 값을 저장하는 것이다. 이 동작은 구조체의 첫 번째 주소가 stack pointer 값을 가리키기 때문에 가능하다. uthread.asm 파일을 통해 uthread.c에서 변환된 assembly들을 확인할 수 있다.

gdb를 이용해 작성한 코드를 단계별로 실행해보며 테스트 해 볼 수 있다.

```
(gdb) symbol-file _uthread
Load new symbol table from
"/Users/kaashoek/classes/6828/xv6/_uthread"? (y or n) y
Reading symbols from
/Users/kaashoek/classes/6828/xv6/_uthread...done.
(gdb) b thread_switch
Breakpoint 1 at 0x204: file uthread_switch.S, line 9.
(gdb)
```

_uthread 프로그램을 실행하기 전에 thread_switch에 breakpoint를 걸고 실행해보고 uthread_switch 함수를 수행하기 전 stack을 확인하라.

```
(gdb) p/x next_thread->sp
$4 = 0x4ae8
(gdb) x/9x next_thread->sp
0x4ae8 :  0x00000000  0x00000000  0x00000000  0x00000000
0x4af8 :  0x00000000  0x00000000  0x00000000  0x00000000
0x4b08 :  0x000000d8
```

next thread 의 stack top에 있는 0xd8 값은 무엇을 의미하는가?

Submit: 수정한 uthread_switch.S

제출 양식: hw8_학번.S